UCRL-CONF-209556

Restructuring Large Data Hierarchies for Scientific Query Tools

Megan Thomas

February 10, 2005

The 9th International Database Engineering and Application Symposium
Montreal, Canada
July 25, 2005 through July 27, 2005

Restructuring Large Data Hierarchies for Scientific Query Tools

Megan Thomas, William Arrighi, Chuck Baldwin, Terence Critchlow, Tina Eliassi-Rad and Susan Hazlett
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
{mthomas, arrighi2, baldwin5, critchlow, eliassirad1, hazlett1}@llnl.gov

Abstract

Today's large-scale scientific simulations produce data sets tens to hundreds of terabytes in size. The DataFoundry project is developing querying and analysis tools for these data sets. The Approximate Ad-Hoc Query Engine for Simulation Data (AQSIM) uses a multi-resolution, tree-shaped data structure that allows users to place runtime limits on queries over scientific simulation data. In this AQSIM data hierarchy, each node in the tree contains an abstract model describing all of the information contained in the subtree below that node. AQSIM is able to create the data hierarchy in a single pass. However, the nodes in the hierarchy frequently have low node fanout, which leads to inefficient I/O behavior during query processing. Low node fanout is a common problem in tree-shaped indices. This paper presents a set of one-pass tree "pruning" algorithms that efficiently restructure the data hierarchy by removing inner nodes, thereby increasing node fanout. As our experimental results show, the best approach is a combination of two algorithms, one that focuses on increasing node fanout and one that attempts to reduce the maximum tree height.

1. Introduction

Scientific investigations increasingly depend upon simulation to supplement traditional experiments. Experiments can be expensive, like automotive crash studies, or impossible, like examining the evolution of stars [9] or of the universe itself [8]. The data generated from one large-scale scientific simulation run can easily reach tens to hundreds of terabytes in size, spread across thousands of files. As data set sizes grow, finding useful information becomes increasingly difficult. At Lawrence Livermore National Laboratory (LLNL), the DataFoundry project research and development efforts for the Approximate Ad-Hoc Query Engine for Simulation Data (AQSIM) [1, 2, 3, 7] focus on querying the scientific data in these collections, enabling scientists to easily find subsets of interesting data.

We take an approximate approach to queries across these data sets. Scientists can place a time constraint on each query, and the AQSIM tool returns the best approximate answer it can within the time constraint. To make time-constrained queries possible, AQSIM uses a multi-resolution, hierarchical representation of the data in which each node in the hierarchy stores a model of all the data in the subtree rooted at that node. Query processing traverses the hierarchy, evaluating the query against the data model at each node in the traversal to determine if the subtree below that node should be traversed or not. If the subtree can satisfy the query, AQSIM prioritizes it for traversal, in order to best use processing time before the query time limit is reached.

Traditional indexing techniques will not work well with these data sets due to the high dimensionality and the sheer size of the data sets. The supercomputers that run the scientific simulations are heavily used and are not available to sort the data sets or build AQSIM hierarchies. Also, sorting the data to facilitate bulk-loading a traditional index like an R*-tree [5] would require storing intermediate copies of the (sorted) data after sorting and before the traditional index is built. Our scientific simulation data sets can easily be large enough that there will not be enough disk space to store multiple copies of the data. In addition, sorting large data sets requires multiple passes over the data. Our aim is to create the AQSIM data hierarchy in one pass.

AQSIM builds the current multi-resolution hierarchy [4] based solely on spatial characteristics of the data and produces a hierarchy that is a tree with a relatively small fanout and large depth – far from ideal

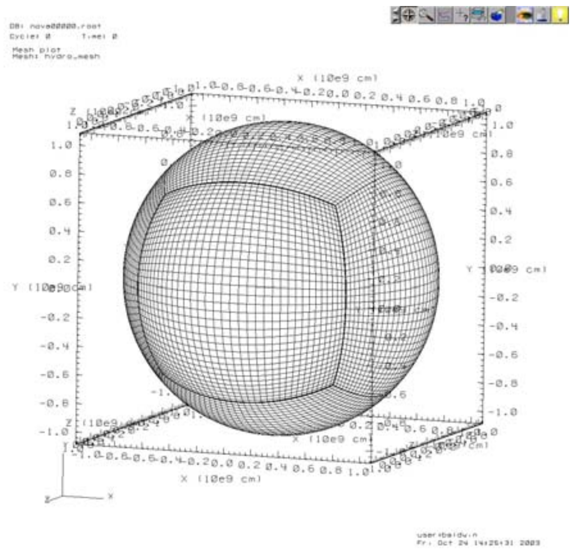

Figure 1: Astrophysics mesh

for a disk-based data structure. In Section 3 we cover the current hierarchy in greater detail.

In this paper we discuss several algorithms for restructuring, or “pruning”, a data hierarchy by deleting carefully selected inner nodes from the hierarchy to create a shorter, higher fanout data structure.

In Section 4, we present several algorithms for selecting good inner nodes for deletion, and priority orderings to use, where necessary, in selecting new parent nodes to “adopt” the orphaned child nodes. While we only demonstrate these algorithms on our hierarchy, the algorithms can be applied to a variety of hierarchical data structures [11, 13].

As shown by our experimental results, presented in Sections 5 and 6, we have found that the best way to restructure the data hierarchy is to take one pass over the tree, eliminating nodes and distributing their children to siblings of the eliminated nodes (updating each sibling’s data model in the process). This increases average node fanout. Then follow the first pass with a second pass, eliminating nodes and moving their children up, to reduce the tree height. The pruning algorithms used in each pass may both be applied to each piece of the hierarchy while the piece resides in memory, making the pruning process effectively one pass from the perspective of the disk-based data structure.

2. Scientific simulation data

Large-scale scientific simulation programs typically produce spatio-temporal data sets in one of several *mesh* data formats, like SILO [18] or HDF [16]. A mesh consists of an interconnected grid of small regions, or *zones*. Each zone in a mesh represents a unique spatial region and contains data (i.e. variables) representing the state of the simulated scientific phenomenon in that region at a specific time.

Meshes may be *regular*, consisting of a single type of zone (such as rectangular boxes), or *irregular*, consisting of arbitrarily shaped zones, depending on the needs of the scientists. For pragmatic reasons, a single mesh may be broken up and spread across hundreds of files, or *domains*. The scientific simulation generating the data defines the domains, which are usually spatially noncontiguous. For our purposes, it is sufficient to think of a domain simply as the chunk of simulation data associated with, for example, a supercomputer node that handled calculations simulating the scientific phenomenon in that domain. (This is a simplification. Further information on mesh research and software is available at [14].) Figure 1 shows a mesh for an astrophysics simulation of a star.

Simulations follow a scientific phenomenon over a span of simulated time, writing out the complete state of the phenomenon at each *timestep*. A timestep may represent a microsecond of activity in a simulation of the mixing of two fluids, or a million years in the evolution of the universe. The mesh itself may also evolve – the spatial regions zones represent may change – over the course of the simulation, in order to maintain the finest mesh granularity at the currently most active areas of the simulated phenomenon. Therefore, each timestep the simulation writes out contains a reference to the associated mesh and also the appropriate variable information for each zone in the mesh. A typical simulation data set will contain data for several hundred timesteps. A more complete description of scientific mesh data, including a contrast with traditional relational data, is provided in [15].

3. Approximate ad-hoc query engine for simulation data

Before we can perform any approximate query on a data set, we need to preprocess the data to create an Ad-Hoc Query Hierarchy (AQH) for each timestep in a data set. AQSIM currently builds the hierarchy using a topological agglomeration method [4] that combines (agglomerates) neighboring zones into larger spatial regions that become the nodes in the hierarchy one level up from the zones, or leaf level. The topological agglomeration algorithm then combines neighboring spatial regions, building the next higher level of nodes in the hierarchy, and continues repeating agglomeration steps until it gathers neighboring regions into the root node of a tree. The algorithm we use is akin to an OctTree generation algorithm [17], but works from the bottom up instead of top down. Our algorithm has been generalized to work with both regular and irregular meshes. As each agglomeration operation is performed, AQSIM creates a statistical model (i.e. minimum, maximum, average, standard deviation) of the data within the new agglomerated region. We have explored several alternative models and agglomeration approaches, but since all resulting AQH hierarchies have similar properties, we do not present the alternative models further here.

Because the agglomeration routine uses neighbor information in selecting which regions to combine, the fanout of the resulting hierarchy is limited to the maximum of the number of neighbors a zone has, which is only eight for a rectangular, three-dimensional grid. After agglomeration creates an initial hierarchy, our pruning algorithms are applied to improve the hierarchy before we write the AQH structure to disk. Our pruning algorithms both increase the hierarchy fanout and reduce the overall data structure size by removing carefully selected internal nodes from the AQH hierarchy. The sequence of operations, from raw data to AQH hierarchy, is depicted in Figure 2.

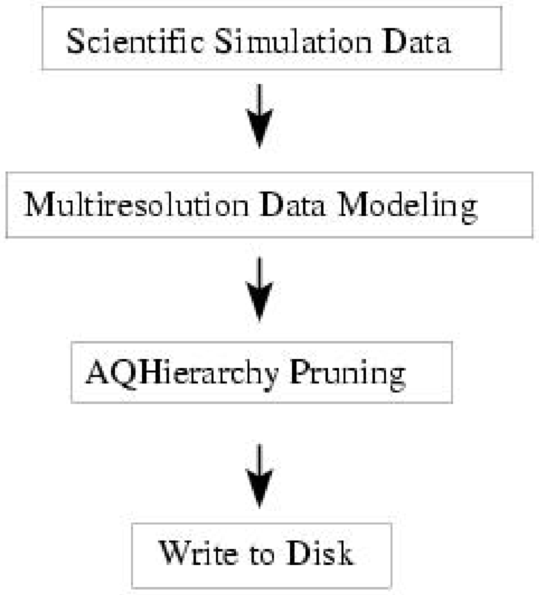

Figure 2: Creating an AQH

After pruning the AQH hierarchy we perform one final optimization before writing the data structure to disk. We eliminate variables stored as part of the data models in child nodes when the variables can be perfectly regenerated using the parent node's data model. For example, if all the data regions contained by the children of a node are at exactly the same temperature, only the parent node will record the temperature data. Due to the constraints for initializing simulations and the physical characteristics of the processes being simulated, this optimization proved to be quite effective. For example, in the White Dwarf star simulation data set described in Section 5 this optimization reduces the AQH size approximately 20%.

Queries over a data set begin at the AQH root node for each of the timesteps in the data set. The query engine identifies nodes (starting with the root nodes) that match the query condition and inserts them into a priority queue. The query engine refines nodes by replacing nodes matching the query with the nodes' children. The priorities of unrefined nodes in the priority queue are based on the query and the accuracy of the data models in the unrefined nodes. When either the query runtime limit expires or the all of the leaf nodes that match the query are identified, the query result is returned. AQSIM writes query results to a new mesh that contains only those zones that meet the query condition. Since each node in the AQH contains a model of the data in the subtree below it, that model may be used to generate an approximate result if there is insufficient time to refine an interior node. Due to the importance of returning conservative estimates to our users, while an approximate result may contain false positives (i.e. regions that do not match the query condition may be returned), the query engine is designed to prevent any false negatives from occurring (i.e. no region that meets the query condition will be omitted).

Table 1: Variable definitions

Variable	**Meaning**
I	dimension of current variable
N	total number of dimensions
K	child node
R	root node
S	sibling node
p	parent node
f^x	fanout of node x
v_i^x	mean value of variable i in node x
$\min_i^x$	minimum of variable i in node x
$\max_i^x$	maximum of variable i in node x

Table 1 defines some variables we use in this paper.

4. Algorithms for pruning data hierarchies

We explored two basic approaches to pruning nodes out of a hierarchy — `Move Up` and `Move Side` — plus a third approach — `Move Both`, is a hybrid of the first two approaches.

All of our pruning algorithms perform a post-order traversal [19] of the hierarchy, pruning from the bottom up. The pruning algorithms follow topological data agglomeration and statistical data model creation algorithms that work post-order, building the hierarchy and constructing the data models from the bottom up, domain by domain. In order to ensure that pruning does not require an additional pass over the data set, the pruning algorithms must also traverse the data post-order, prune bottom up and operate on only the information available within a given partition of the data set, usually a domain. However, due to the speed of memory access, we allow the pruning algorithms to make multiple passes over the data partition currently in memory to improve the associated tree characteristics. Since the data is in memory during the multiple passes, the extra passes do not cause extra disk I/Os.

When deleting internal nodes from a hierarchy, the crucial questions are which nodes to delete and where to re-attach their child nodes. When developing the pruning algorithms, we focused on moving orphaned nodes to the parent node (`Move Up`) or to an appropriate sibling node (`Move Side`) because these approaches are local operations – no global knowledge regarding the tree structure or data set is necessary.

Gathering global knowledge would require at least one more full pass over the entire data set. For example, working explicitly to balance the height of the AQH would require knowledge of the total number of nodes and the height of the tree before node deletion and movement decisions could be made. Our pruning algorithms require no such *apriori* knowledge.

We specify the desired maximum and minimum fanouts for AQH nodes as input to the pruning process. At present, we are most interested in how well the algorithms achieve the specified fanout.

The pruning algorithms pass over the section of the AQH currently in memory multiple times if the initial pass does not increase the minimum node fanout to the requested value. The pruning algorithms move on to the next section of the AQH when the requested minimum node fanout has been achieved or when no pruning can be performed without overrunning the requested maximum node fanout.

Unlike our plant pruning analogy, we do not ever eliminate leaf nodes from the data structure. The leaf nodes store the finest granularity data from the original scientific simulation mesh. Future versions of the AQH may eliminate leaf nodes, vastly decreasing the size of the AQH storage files. However, for now, we store the leaf nodes so that the scientists can get the same results from queries over the AQH and fully scanning the original data.

The pruning algorithms we present in this paper are not restricted to any one data modeling algorithm or to the AQH alone. The pruning algorithms can be applied to any hierarchy where the data in each node in the hierarchy summarizes the data in all the nodes below that node, and there are no explicit restrictions on the heights of subtrees. For example, the pruning algorithms can work on a QuadTree [17], but not a height-balanced R*-tree [5].

4.1 Moving nodes up

The `Move Up` pruning algorithm eliminates child nodes and moves the grandchildren nodes up a level to become immediate children of the former grandparent, which becomes parent. The children of eliminated nodes *move up* in the hierarchy. This operation both increases the fanout of the original grandparent node, and potentially reduces the height of the subtree below the original grandparent node.

For each non-leaf node, *p*, encountered during a pass over the current section of the AQH, `Move Up` sorts *p*’s non-leaf children using one of the priority orderings listed below. `Move Up` then deletes the highest-priority child node, *k*, and adds *k*'s children to *p*'s set of children. `Move Up` continues deleting *p*’s children until it cannot find a child of *p* whose deletion will **not** push the fanout of *p* above the specified maximum fanout. `Move Up` does not eliminate any of *p*’s grandchildren that have moved up during the current pass.

When only one child node, *k*, exists such that ($f^k + f^p - 1$) ≤ *maximum_fanout*, then *k* is selected for deletion. However, if there is more than one child node that satisfies this fanout restriction and deleting all of the child nodes would increase the parent node’s fanout above the allowed maximum, then the child nodes must be prioritized for deletion. Once the child nodes have been prioritized, `Move Up` deletes the child nodes in priority order until no child nodes remain whose deletion would not overrun the specified maximum fanout.

We explored several prioritization orderings for `Move Up` pruning. The descriptions below refer only to non-leaf child nodes whose deletion will not push the parent node's fanout above the maximum fanout threshold.

- `Random`: Order child nodes randomly.
- `Darwin`: Order child nodes by their fanout. Eliminate the child with lowest fanout. Use distance of child nodes from their farthest leaf node descendants as a tie-breaker; when more than one child has the lowest fanout, eliminate the child with the longest path to a leaf node first.
- **Wavelet inspired:** Order child nodes by how much the data variables they store vary relative to the parent node's data variables. Unlike the first three `Move Up` prioritization orderings, the wavelet-inspired orderings take the values of the data stored on each node into consideration when prioritizing nodes for elimination. For these priority orderings, we include the spatial coordinates of the nodes as three more data dimensions. We use sums over the data dimensions rather than products because the data values in some dimensions are frequently constant over large swathes of the data set. Since subtracting the constant value from itself yields zero, multiplying instead of adding would make the resulting priorities always zero for many nodes.
  - `Wavelet 1`: Order child nodes by the sum of the differences of their mean data variable values from the mean data variables in the parent node. Normalize the differences using the variable ranges over the entire data set, which are stored in the root node, *r*. Eliminate the node with the largest difference first. Note that this priority ordering may not be used in one-pass pruning unless the scientific simulation data format includes summary information for the entire data set, which is not usually the case with scientific simulation mesh data, but may be so for other data types.

$$\sum_{i=0}^{n} \frac{|v_i^p - v_i^k|}{\max_i^r - \min_i^r}$$

  - `Wavelet 2`: Order child nodes by how large their data ranges are, relative to *p*'s data ranges. Eliminate the child node with the widest relative ranges first, the "biggest" child.

$$\sum_{i=0}^{n} \frac{\max_i^k - \min_i^k}{\max_i^p - \min_i^p}$$

  - `Wavelet 3`: Order child nodes by the differences of their mean data variable values from those in the parent node, normalized by the ranges of the data variables in *p*. Eliminate the node whose mean data variable values are farthest from the parent node's mean data values first. Like `Wavelet 1`, this ordering aims to eliminate the "most different" child.

$$\sum_{i=0}^{n} \frac{|v_i^p - v_i^k|}{\max_i^p - \min_i^p}$$

4.2 Moving nodes sideways

Like the `Move Up` pruning algorithm, the `Move Side` pruning algorithm performs a post-order traversal of the given hierarchy. The `Move Side` pruning algorithm uses `Move Up` priority orderings for selecting the node, *e*, to eliminate from the set of children of the current node. Then `Move Side` uses one of the priority orderings listed below to select the node, *s*, a sibling of *e* that will "adopt" *e*'s children. The children of eliminated node *e move side*ways to become children of *s*. The sibling nodes in the priority orderings below are those whose fanouts will not be pushed over the maximum fanout by the addition of eliminated node *e*'s child nodes.

- `Random`: Pick *s* randomly.
- `Darwin`: Give *e*'s children to sibling *s* that has the lowest fanout, the fewest children.
- `Euclid`: Recall that every zone and node in the simulation mesh has spatial *x*, *y* and *z* coordinates, in addition to the data variables. Give *e*'s children to the sibling whose Euclidean distance from *e* is smallest. If global information on spatial dimension ranges is available for data sets with non-symmetric spatial dimensions, it may be best to normalize the spatial distances on each dimension. Because our test data sets were roughly symmetric (stellar spheres), we did not normalize the Euclidean distances.

Table 2: Experiments

Pruning Algorithms	Pruning Priority Algorithms			Data Sets	Fanout Ranges
	`Move Up`	`Move Side`			
	Node Elimination Prioritizing Algorithms	Node Elimination Prioritizing Algorithms	Sibling Prioritization Algorithms		

No pruning
`Move Up`
Node Elimination
`Move Side`
Node Elimination,
`Move Both`
Node Elimination
	`Random`
	`Darwin`
	`Wavelet 1`
	`Wavelet 2`
	`Wavelet 3`
		`Random`
		`Darwin`
		`Wavelet 2`
			`Random`
			`Darwin`
			`Euclid`
			`Wavelet A`
			`Wavelet B`
			`Wavelet C`
				White Dwarf
				Shock
				Mid-Life Star
					10 – 20
					20 – 40
					40 – 60

- `Wavelet A`: Order sibling nodes by the differences of their mean data variable values from *e*'s mean data variable values, normalized by the ranges of the data variables in the entire data set, which are stored in root node, *r*. Give *e*'s children to the nearest sibling. Note that this priority algorithm requires global information, therefore it may not be suitable for one-pass pruning.

$$\sum_{i=0}^{n} \frac{|v_i^e - v_i^s|}{\max_i^r - \min_i^r}$$

- `Wavelet B`: Order sibling nodes by the differences of their mean data variable values from *e*'s means, normalized by the ranges of the variables in their parent, *p*. (`Wavelet B` is similar to `Wavelet A`, but does not require global information.)

$$\sum_{i=0}^{n} \frac{|v_i^e - v_i^s|}{\max_i^p - \min_i^p}$$

- `Wavelet C`: Order siblings, *s*, by the overlap of their data value ranges with *e*’s data value ranges, normalized by the range of the parent node *p*’s data value ranges. *High* is the lowest value for $\max_i$ between *e* and *s*; *low* is the highest value for $\min_i$ between *e* and *s*. If *low* is greater than *high*, there is no overlap between the two nodes in data dimension *i*. In that case, we add zero to the sum, not the (negative) value of the fraction.

$$\sum_{i=0}^{n} \frac{high_i - low_i}{\max_i^p - \min_i^p}$$

4.3 Moving nodes both ways

We also ran experiments that combined the best of our `Move Up` and `Move Side` pruning approaches. For the section of the hierarchy currently in memory, `Move Both` iterates over the nodes, pruning using `Move Side`. `Move Both` then iterates again over the nodes, using `Move Up`. `Move Both` moves children of eliminated nodes *both* sideways and up. Favorable results were not achieved in experiments using `Move Up`, followed by `Move Side`, so we do not present the results of those experiments here.

5. Experimental data sets and parameters

We ran our experiments over three scientific simulation data sets made available by LLNL physicists. In each experiment we built and pruned an AQH for each timestep in each data set. The White Dwarf data set contains 22 timesteps of a simulation of a star exploding, totaling approximately 3.2 GB, roughly 144 MB per timestamp. The White Dwarf data set had 29 data dimensions. Figure 3 shows a graph of one variable in the last timestep in the White Dwarf data set. The Shock data set is one timestep of a shockwave propagating through material, approximately 6.8 GB. The Shock data set has 37 data dimensions. The third data set, Mid-Life Star, is 16 timesteps simulating a star in the middle of its life and totaling about 6.5 GB of data — roughly 404 MB per timestamp. The Mid-Life Star data set has 27 data dimensions.

In the figures in Section 6, the displayed results for each data set are averages over all of that data set’s timesteps.

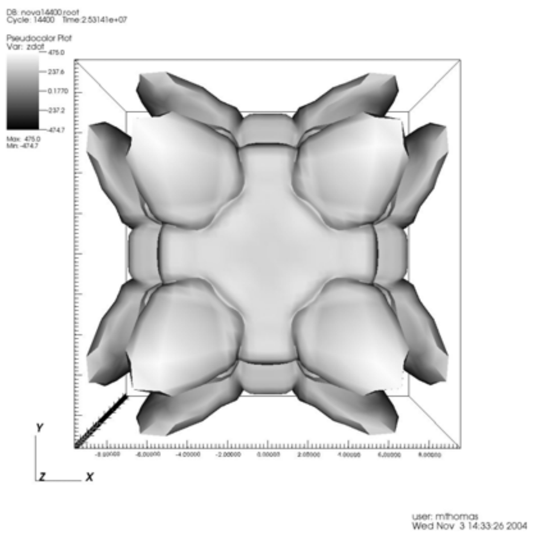

Figure 3: White dwarf data set, a simulation of a star exploding

The requested minimum and maximum fanout values we experimented with are low relative to the fanouts a B-tree (one dimension) or spatial R-tree (two or three dimensions) uses. However, at each node the AQH stores a statistical data model for the subtree (minimum, maximum, average and standard deviation for every data dimension) below that node. Therefore, even at our relatively low experimental fanouts, a set of child nodes (read and processed as a unit during queries) spans multiple disk pages, giving a reasonable balance between node size and number of disk pages retrieved per access to the AQH on disk.

For each combination of data set, requested minimum/maximum fanout range, and `Move Up` node elimination priority algorithm listed in Table 2, we created an AQH and then pruned the hierarchy. For the `Random` priority orderings, we created and pruned an AQH multiple times with different starting seeds for the pseudo-random number generator and averaged the results.

Each `Move Side` experiment was a combination of:

- A data set
- A requested minimum/maximum fanout range
- A node elimination ordering, from the list in Table 2, used to select nodes, *e*, for elimination
- A sibling prioritization ordering, from the list in Table 2, used to select the sibling of *e* that will adopt the *e*'s children

For each possible combination of the four parameters listed, we created an AQH and pruned it. If either the node elimination or the sibling prioritization ordering was `Random`, we ran that experiment multiple times with varied starting seeds for the pseudo-random number generator and averaged the results.

Finally, we combined the best `Move Side` and `Move Up` priority algorithms for our `Move Both` pruning experiments.

6. Experimental results

In this section we present the observations that led us to implement `Move Both`, our best overall hierarchy pruning approach, and comment on the structural effects that the various pruning algorithms and node priority orderings have on hierarchies.

In legends for Figures 4-11, when the `Move Side` algorithm is listed in the legend, the first following priority ordering is the node elimination ordering, followed by a '/' and then the sibling selection priority ordering. For example, "`MoveSide Wavelet 2 / Wavelet C`" in the legend means the `Wavelet 2` ordering was used to prioritize nodes for elimination, and the `Wavelet C` ordering was used to decide which sibling adopted an eliminated node's children. When the `Move Both` algorithm is listed in the legends, the first following priority ordering is the one used to order nodes for elimination in both the `Move Up` and `Move Side` passes and the priority ordering listed after the '/' is the sibling node prioritization ordering used in the Move `Side` pass.

6.1 Pruning effects on average node fanout

Figures 4, 5 and 6 show that the `Move Side` algorithm more rapidly improves average node fanout than the `Move Up` algorithm. The `Move Side` algorithm is closer to the maximum requested for 10 – 20 (Figure 4) and 20 – 40 (Figure 5) fanouts, though the `Move Up` pruning algorithm eventually catches up (Figure 6).

`Move Side`'s more rapid fanout improvement is due to how it moves nodes as it prunes. As sibling nodes are deleted and their children are transferred, the parent nodes' fanouts decrease. This creates more opportunities for node deletion at the next level closer to the root node. Recall that the pruning algorithms use post-order hierarchy traversal; child nodes are pruned before their parents. A node whose fanout is initially too high to accept another block of child nodes may have its fanout reduced to the point where it can take custody of nephew nodes without overrunning the specified maximum fanout. However, `Move Side` is limited by the fact that on any given inner node it eventually reaches a point where it may have a child node eligible for elimination but no available sibling of that child to adopt the eliminated node's children. `Move Up` does not suffer from this liability, which accounts for why it surpasses `Move Side`'s average fanout improvements when the requested maximum fanout is high enough.

Figures 4, 5, 6 and 7 also show that the choice of node elimination priority ordering for `Move Side` has negligible impact upon the performance of the `Move Side` algorithm overall —`Darwin`, `Random` and `Wavelet 2` all behaved the same. Shortly we will show that sibling node choice does matter.

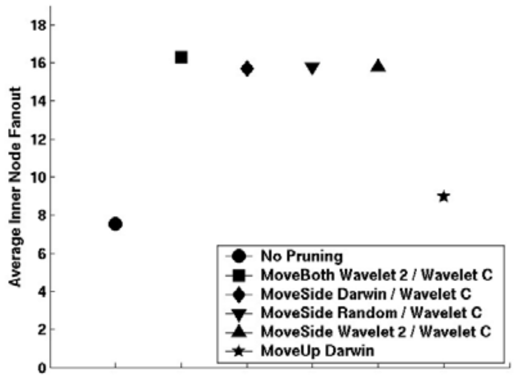

Figure 4: Effects of pruning on average node fanout for the White Dwarf data set and fanout range from 10 to 20. Only the best node elimination priority orderings have been displayed.

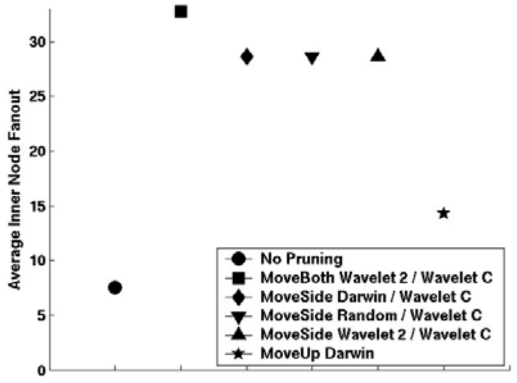

Figure 5: Effects of pruning on average node fanout for White Dwarf data set and fanout range from 20 to 40. Only the best node elimination priority orderings only have been displayed.

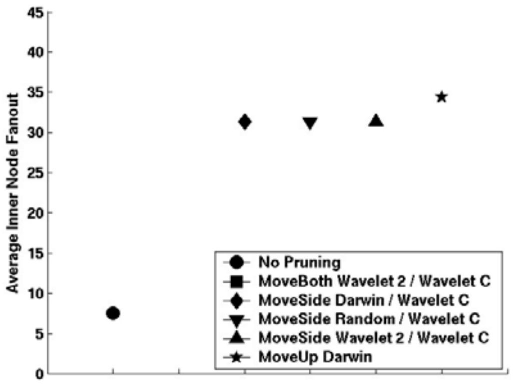

Figure 6: Effects of pruning on average node fanout for White Dwarf data set and fanout range from 40 to 60. Only the best node elimination priority orderings have been displayed.

Figure 7 shows that the `Wavelet C` sibling node selection ordering improves average node fanout better than any of the other `Move Side` orderings for selecting a sibling node to receive orphaned child nodes, regardless of which node elimination ordering is used. The node `Move Side` pruning eliminates matters much less than where the eliminated node's children are reattached.

For figures after Figure 7 with `Move Side` results, where the node elimination ordering is not specified in the legend `Wavelet 2` was used; the results for other node elimination priority orderings are entirely consistent.

The topological data modeling algorithms we developed are very effective at building AQH nodes with fanouts of precisely eight. The lack of strong performance differences between node elimination priority orderings, particularly evident in Figure 7, may be partially attributable to the even node fanouts in the AQH before pruning begins. However, any pruning operations on a hierarchy built over an OctTree or similar structure will also encounter homogenous node fanouts. In the next subsection we will decide which node elimination algorithm is best based, not upon node fanout effects, but other hierarchy metrics.

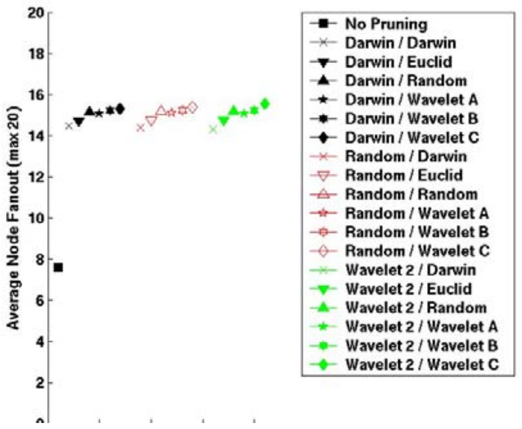

Figure 7: Average node fanout for Move Side pruning over shock data set with fanout range from 10 to 20. Wavelet C is the best of the sibling priority orderings.

6.2 Other pruning effects

By its nature, the `Move Side` algorithm cannot decrease the height of the hierarchy. The `Move Side` algorithm move nodes only sideways — never up. A shorter hierarchy is desirable because a shorter path from root to leaves means a smaller worst-case number of disk I/Os between root node and leaves. As Figure 8 shows, among the `Move Up` node elimination priority orderings, `Darwin` and `Wavelet 2` most consistently reduce AQH height. This is not surprising for `Darwin`, since it explicitly uses hierarchy height to make node elimination decisions. `Wavelet 2` eliminates the child nodes whose data ranges are largest, relative to the parent; that the "largest" child nodes are most likely to have the longest subtrees is intuitively logical.

In Figure 9 we look at the sum of the normalized node perimeters. We use the sum of $(\max_i – \min_i)/(\max_r – \min_r)$ for all dimensions, over all inner nodes i in a timestep's AQH – the sum over the node perimeters, normalized using the global minimum and maximum for each dimension. This is a rough measure of the amount of space in the data space that the inner nodes cover — or how "big" the inner nodes are. We average this perimeter metric over all the timesteps in the data set. We prefer that the normalized node perimeters be small since unnecessarily large inner nodes may lead to queries traversing subtrees in which no data relevant to the query will be found. (R*-tree indexes [5] minimize node perimeters when splitting nodes for the same reason.) Figure 9 shows perimeter rather than volume because our data sets include dimensions, j, where $(\max_j – \min_j)$ is zero for many of the nodes in the AQH, so the volume for most of the nodes would also be zero.

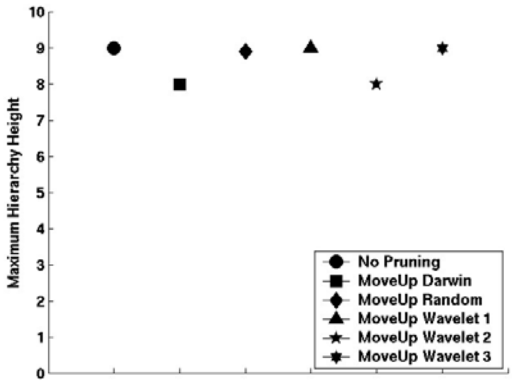

Figure 8: Effects of pruning on maximum hierarchy height for White Dwarf data set and fanout range from 40 to 60.

As Figure 9 shows, the `Move Up` priority ordering `Wavelet 2` performs best at reducing the average inner node perimeters. Among the `Move Side` sibling node orderings, the `Wavelet A` and `Wavelet C` sibling selection orderings are more effective than the others. Recall that `Wavelet A` uses some global information in making decisions; `Wavelet C` does not.

Because it does well at reducing node perimeters and uses no global information to do so, we consider `Wavelet C` to be the best sibling node selection priority ordering. `Wavelet 2` performs well at both reducing AQH height and at minimizing node perimeters, so `Wavelet 2` is the best node elimination priority ordering.

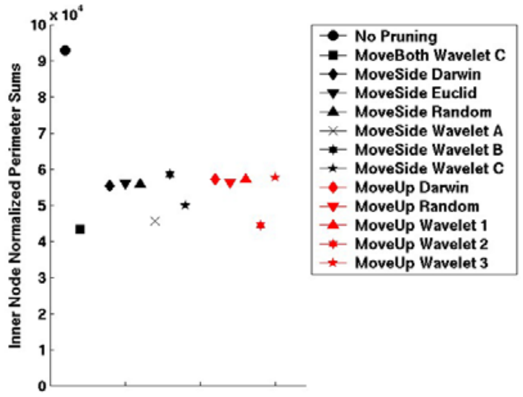

Figure 9: Effects of pruning on node data perimeter for White Dwarf data set and fanout range from 20 to 40. Move Side and Move Both are using the Wavelet 2 node elimination priority ordering. Wavelet 2, Wavelet A and Wavelet C orderings do best at reducing the node perimeters.

Figure 10 shows that, even though pruning does not make a large difference in AQH maximum height, most of the nodes in the AQH decrease in height significantly. The leftmost peak in the graph is the height the leaf nodes reach if all nodes in the tree are at the maximum requested fanout for that experiment.

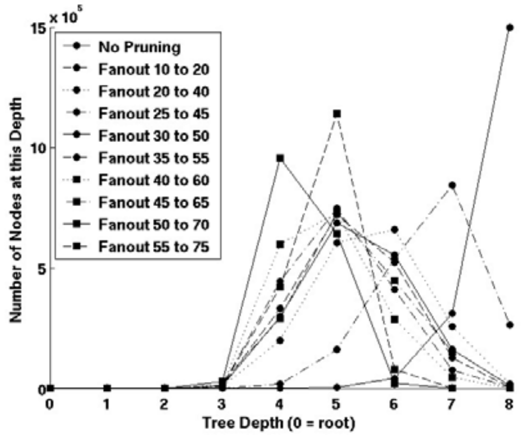

Figure 10: Histogram of number of leaf nodes vs. distance from root node in hierarchies created using Move Up pruning with the Wavelet 2 node elimination priority ordering over the Mid-Life Star data set. Depth zero is the root node. Notice how Move Up pruning "pulls" the nodes up towards root.

Figure 11 shows the achieved maximum fanout compared to the requested maximum fanout. In all but one test the requested minimum fanout was 20 less than the requested maximum. For the maximum fanout of 20, we requested a minimum of 10 in order to ensure that some pruning occurred. (Recall that, in an unpruned AQH, most nodes have a fanout of eight.) Of note in Figure 11 is the relative smoothness of the achieved fanouts, indicating that the pruning algorithms are not overly sensitive to the input minimum and maximum fanouts. (Figure 11 shows results from `Move Up` experiments; `Move Both` and `Move Side` results were very similar.)

Figures 4, 5, 6 and 9 all include the results for `Move Both` experiments that use the `Wavelet 2` node elimination priority algorithm and the `Wavelet C` sibling selection algorithm. Note that in all cases, `Move Both` clearly outperforms both simple `Move Up` and `Move Side.`

Note that one reason the average fanout achieved can be smoothly specified is that the pruning algorithms have a steady supply of low fanout nodes from the topological node agglomeration algorithm to work with. Once that supply is exhausted, the only nodes available to prune will already have a fanout near 64 (8^2, where 8 is the maximum fanout produced by the topological agglomeration algorithm). Our algorithms are limited in the ability to achieve precise maximum fanouts, at higher requested fanouts, by the fact we only move sets of child nodes as a unit. Simple alterations to the pruning algorithms would make it possible to achieve precisely the requested maximum fanout, if necessary.

For example, pruning algorithms could consider grandchildren for moving up at the same time that they consider child nodes; or pruning could move some child nodes if there is space at parent/sibling receiving node, even if there isn't space to move all the child nodes. Since achieving fanouts within a reasonable range is sufficient for our current needs, we did not implement these changes. Also, the alterations would slow the pruning of each node. A similar effect would occur again if we were to request maximum fanouts above 64^2, but query processing would frequently retrieve irrelevant nodes in an AQH with such large fanouts.

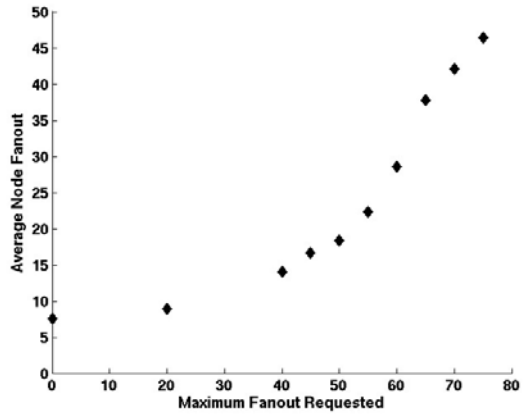

Figure 11: Average node fanout for a range of requested maximum fanouts — Shock data set, Move Up pruning with Wavelet 2 priority ordering. Note the smoothness of the curve indicating that the pruning algorithms are not over-sensitive to their input parameters.

7. Related work

Other approaches to improving the performance of a tree-shaped data access method are to sort and bulk-load data [10, 12] or to develop improved insertion algorithms [6]. While we do build our data structure ahead of its use in queries, as in bulk-loading, we do not require that the data be sorted. Bulk-loading requires that the data be sorted into leaf node sized chunks, then the tree built, in order to achieve good query performance. Sorting requires multiple passes over the data set, but we need to avoid multiple passes over our data sets. Rather than organizing the data, then building a hierarchy, we approach the problem from the other direction by building the hierarchy, then reorganizing it for better performance.

Like our pruning, [21] reorganizes an existing tree structure, focusing on improving poorly organized parts of a tree-shaped data structure. However, the algorithms in [21] are restricted to B-trees, and they actually move data in poorly performing parts of the B-tree around on disk while other operations run concurrently. We only move links between nodes and need not worry about concurrency because our data sets are static.

Other multi-resolution, tree-like data access methods that would be compatible with our pruning algorithms have been proposed, such as [11, 13, 20].

The MRA-Tree [11] was designed specifically with aggregate queries — queries that ask for sum, count, minimum, maximum and other aggregate information — in mind. Their focus is on maintaining the aggregate statistics in a multi-resolution tree during insertion and deletion operations. They do not require a height-balanced structure.

STING [20] is also a hierarchical, space-based data partitioning approach to spatial data mining. Like our AQH, they allow many queries to be answered using statistical information stored at inner nodes, in order to reduce or eliminate the need for all queries to proceed all the way to the leaf level.

A pyramid data structure for browsing Earth science data is introduced in [13]. Like the AQH, their pyramid stores multi-resolution, statistical summaries of finer-grained data at each cell (node) and can return approximate answers to user queries. The focus of [13] is on populating a pyramid (hierarchy) with statistical summaries and processing queries. They make few assumptions about the structure of the pyramid, so their work could be compatible with our pruning algorithms.

8. Conclusion

We presented three one-pass algorithms for reducing the height and increasing the fanout of a given multi-resolution, tree-shaped hierarchy. We experimented with all three algorithms using the Ad-Hoc Query Hierarchy (AQH), our data structure for approximate querying with query runtime limits. Our `Move Side` hierarchy pruning algorithm removes inner nodes and moves their children over to carefully chosen sibling nodes. `Move Up` removes inner nodes and moves their children up to the removed node's parent. We found that our hybrid, `Move Both` algorithm, which uses `Move Side` and then `Move Up`, achieves the best overall effect on the hierarchy. `Move Both` performance benefits from `Move Side`'s superior effect on inner node fanout, and `Move Up`'s good effects on hierarchy height.

Because the scientific simulation data sets that the Approximate Ad-Hoc Query Engine for Simulation Data expects to handle are so large, we worked to avoid multiple passes over the data while creating our data hierarchies. Hence, our tree-shaped hierarchy pruning algorithms are all one-pass, working on local pieces of the tree structure without needing knowledge of the global hierarchy characteristics.

We found that our `Move Up` algorithm reduced hierarchy height, though the `Move Side` algorithm was better at achieving the requested node fanouts. We favor eliminating child nodes based on how completely they overlap their parent node's data range (`Wavelet 2` node priority ordering and `Wavelet C` sibling priority ordering) because prioritizing nodes thus delivered the best results in our experiments over several real data sets. These prioritization orderings also require no global knowledge to order nodes.

`Move Both` is, however, the best pruning algorithm, iterating over each local piece of the hierarchy twice — the first time applying `Move Side` and the second time `Move Up`. It achieves average node fanouts even closer to the maximum requested fanout than `Move Side` while also reducing the maximum height of the AQH as well as `Move Up` pruning does.

For future work we could investigate splitting the children of an eliminated node up among multiple sibling nodes during `Move Side`, when no single sibling has sufficient space to accept all the orphaned children and not overrun the desired maximum fanout. We plan to investigate integrating tree structures generated by other data modeling algorithms, which cluster data based upon all the non-spatial information, with the tree-shaped AQH, producing and pruning a queryable tree or directed acyclic graph.

9. Acknowledgments

We thank Jack Reaugh and the LLNL/IGPP Djehuty astrophysics project for contributing scientific simulation data for our experiments. We thank Gabrielle Rennie for valuable editorial advice.

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405 ENG-48.1.

10. References

[1] G. Abdulla, C. Baldwin, T. Critchlow, R. Kamimura, Lozares, R. Musick, N.A. Tang, B. Lee, and R. Snapp. “Approximate Ad-Hoc Query Engine for Simulation Data,” Proc. *of the First ACM+IEEE Joint Conf. on Digital Libraries (JCDL)*, ACM Press, 2001, pp 255-256.

[2] G. Abdulla, T. Critchlow and W. Arrighi. “Simulation Data as Data Streams,” *SIGMOD Record*, 33(1): 89-94, March 2004.

[3] C. Baldwin, T. Critchlow, and G. Abdulla. “Multi-Resolution Modeling of Large-Scale Scientific Simulation Data,” Proc. *of the 12th ACM International Conference on Information & Knowledge Management*, ACM Press, 2003, pp 40-48.

[4] C. Baldwin, T. Eliassi-Rad, G. Abdulla, and T. Critchlow. “The Evolution of a Hierarchical Partitioning Algorithm for Large-Scale Scientific Data: Three Steps of Increasing Complexity,” *Proc. 15th International Conference on Scientific and Statistical Database Management*, IEEE Computer, 2003, pp 225-228.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles,” *Proc. of the ACM-SIGMOD International Conference on Management of Data*, ACM Press, 1990, pp 322-331.

[6] R. Choubey, L. Chen, and E. Rundensteiner. “GBI: A Generalized R-tree Bulk-Insertion Strategy,” *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, 2002, pp 488-494.

[7] T. Eliassi-Rad, T. Critchlow, and G. Abdulla. “Statistical Modeling of Large-Scale Simulation Data,” *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, 2002, pp 209-226.

[8] A. Hellemans and M. Mukerjee. “Computing the Cosmos,” *IEEE Spectrum*, August 2004, pp 28-34.

[9] A. Heller. “Building a Virtual Telescope,” *Science & Technology Review*, May 2002. UCRL-52000-02-5. http://www.llnl.gov/str/

[10] I. Kamel and C. Faloutsos. “On Packing R-trees,” *Proc. of the International Conference on Information and Knowledge Management*, ACM Press, November 1993, pp 490-499.

[11] I. Lazaridis and S. Mehrotra. “Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure,” *Proc. of the ACM-SIGMOD International Conference on Management of Data*, ACM Press, 2001, pp 401-412.

[12] S. T. Leutenegger, M. A. Lopez, and J. Edgington. “STR: A Simple and Efficient Algorithm for R-tree Packing,” *Proc. of the 12th International Conference on Data Engineering*, IEEE Computer Society, April 1997, pp 497-506.

[13] Z. Li, X. Wang, M. Kafatos, and R. Yang. “A Pyramid Data Model for Supporting Content-Based Browsing and Knowledge Discovery,” *Proc. of the 10th International Conference on Scientific and Statistical Database Management*, IEEE Computer Society, July 1998, pp 170-179.

[14] Meshing Research Corner, Steve Owen: http://www.andrew.cmu.edu/users/sowen/mesh.html. Also, Mesh Generation & Grid Generation on the Web, Robert Schneiders: http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html.

[15] R. Musick and T. Critchlow. “Practical Lessons in Supporting Large-Scale Computational Sciences,” *Proc. of SIGMOD Record*, Vol 28, No. 4, ACM Press, 1999, pp 49-57.

[16] NCSA Hierarchical Data Format. http://hdf.ncsa.uiuc.edu/

[17] H. Samet. “The Quadtree and Related Hierarchical Data Structures,” *ACM Computing Surveys*, Vol 16, No. 2, ACM Press, 1984, pp 187-260.

[18] *SILO User's Guide* , UCRL-MA-118751. http://www.llnl.gov/bdiv/meshtv/manuals.html

[19] G. Valiente, *Algorithms on Trees and Graphs*, Springer-Verlag, Berlin, 1998.

[20] W. Wang, J. Yang and R. Muntz. “STING: A Statistical Information Grid Approach to Spatial Data Mining,” *Proc. of the 23rd International Conference on Very Large Databases*, Morgan Kaufmann, August 1997, pp 186-195.

[21] C. Zou and B. Salzberg. “Safely and Efficiently Updating References During On-Line Reorganization,” *Proc. of the 24th International Conference on Very Large Databases*, September 1998, pp 512-522.